PyConTurb: an open-source constrained turbulence generator

Rinker, Jennifer M.

Published in:
Journal of Physics: Conference Series

Link to article, DOI:
10.1088/1742-6596/1037/6/062032

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

PyConTurb: an open-source constrained turbulence generator

IOP Conf. Series: Journal of Physics: Conf. Series **1037** (2018) 062032 doi:10.1088/1742-6596/1037/6/062032

PyConTurb: an open-source constrained turbulence generator

Jennifer M. Rinker

Technical University of Denmark, Department of Wind Energy, Frederiksborgvej 399, Building 114, 4000 Roskilde Denmark

E-mail: rink@dtu.dk

Abstract. This paper presents an open-source tool that can be used to simulate turbulence boxes constrained by measured data, which is useful for wind turbine model validation. The tool, called PyConTurb for "Python Constrained Turbulence", uses a novel algorithm based on the Kaimal Spectrum with Exponential Coherence method, and the algorithm can efficiently generate turbulence boxes under a wide variety of measurement constraints. The theoretical background for the technique is presented along with a few notes on its implementation in Python. The utility of PyConTurb is demonstrated using real data measured using three-dimensional sonic anemometers at the Denmark Technical University Risø campus. The presented results demonstrate that PyConTurb can successfully generate turbulence boxes from real measured data, including recreating the desired spatial coherence relationships between the simulated and measured time series. PyConTurb is shown to be a promising tool for investigating new spatial coherence models and for future one-to-one wind turbine validation studies.

1. Introduction

The simulation of constrained turbulence boxes is an essential aspect of one-to-one wind turbine validation. A large amount of time and effort can be expended on tuning a wind turbine model, but if we are unable to adequately identify or model the turbulence entering the rotor, then it will be impossible to determine whether simulation-measurement discrepancies arise from errors in the wind turbine model or from the uncertainty in the wind field.

There are a few techniques previously proposed to address this issue in generating turbulence for one-to-one model validation. First, a very simple solution is to use a standard stochastic turbulence simulator (e.g., the Mann turbulence model [4] or the Kaimal Spectrum with Exponential Coherence [5], hereafter abbreviated as KSEC) and to fit the necessary simulation parameters to your measured time series as chosen. For example, we could use the recorded mean wind speed and turbulence intensity but the default parameters from IEC 61400-1 [5] for the other parameters. Alternatively, we could attempt to fit all necessary simulation parameters to the measured time series. Neither of these methods, however, constrain the turbulence box so that it follows the general trends in the measurements. Thus, any unique turbine response caused by a specific phenomenon in the input will not necessarily be recreated in simulation, which is a huge drawback for one-to-one validation applications.

The solution to this issue is to constrain the turbulence simulation in some way so that it is similar to the measurements. Two main techniques have been proposed to accomplish this. First, in the alpha version of TurbSim version 2.0—the open-source turbulence simulation from

the National Renewable Energy Laboratory (NREL) that is based on the KSEC technique—it is possible to provide a user-defined time series to simulate a turbulence box. TurbSim then determines the spectral magnitudes via linear interpolation from the measurements and it correlates the simulated phases to a user-selected reference time series. To the author's knowledge, no tests or verifications of the code have been made public, but it is likely a very useful tool for single-point measurements.

Often, however, we have multiple measurements across a rotor plane. In this case, it may be more appropriate to use the turbulence constraint technique based on the Mann turbulence model [2, 3]. This method generates a random, unconstrained Gaussian field and then superimposes a second Gaussian field that was generated such that the constrained points take on the required values. Unfortunately, there are a few drawbacks to this technique. First, it is computationally more complex than a method based on KSEC. Second, the generated field can sometimes be subject to unrealistic speeds at the unconstrained points due to overfitting. Lastly, there is no implementation of this tool that is publicly available for the wind energy research community.

This paper presents an open-source tool to efficiently generate turbulence boxes constrained by measurements at multiple points and in multiple directions. The tool uses a novel algorithm based on the KSEC simulation technique, and it enforces spatial coherence relationships between all measured and simulated points. The open-source tool, which is called "PyConTurb" for "Python Constrained Turbulence", is currently available in the following git repository: https://gitlab.windenergy.dtu.dk/rink/pyconturb. The objective of this paper is to present the theory behing the simulation algorithm and to demonstrate some of PyConTurb's abilities via a case study.

The remainder of the paper is organized as follows. the theoretical background and computational implementation is discussed in Sec. 2. A case study is presented in Sec. 3 that uses meteorological mast ("met mast") data from the Denmark Technical University (DTU) Risø campus to generate constrained turbulence boxes. Discussion on the flexibility and utility of PyConTurb and future key developments are presented in Sec. 4. Lastly, the paper is concluded in Sec. 5.

2. Methodology

This section contains the theoretical background of the proposed constrained KSEC technique as well as a few notes on its computational implementation in PyConTurb. Both items are discussed in respective subsections below. Further details on PyConTurb, the source code, and examples of how to use the tool can be found at the following GitLab repository: https://gitlab.windenergy.dtu.dk/rink/pyconturb.

2.1. Theoretical background

As mentioned in Sec. 1, the proposed constrained simulation methodology is based on the KSEC method, which is one of the two turbulence simulation techniques in the third edition of the wind turbine design standard IEC 61400-1. In the KSEC method, the turbulence generation occurs in the Fourier domain, where the Fourier magnitudes are deterministic and specified according to a provided Kaimal power spectrum. The phases are randomly sampled, and then the phases corresponding to the longitudinal (i.e., along-wind) turbulence component are correlated using the Cholesky decomposition of the coherence matrix, whose entries are calculated from the exponential coherence function provided in the standard.

The KSEC method is computationally fast and does not suffer from the memory problems associated with the Mann turbulence model, which has lent itself to implementation into open-

source tools such as TurbSim.[1] However, it does not follow the conservation of mass, as the Mann model does; moreover, the standard KSEC model defined in IEC 61400-1 does not feature coherence in the lateral and vertical turbulence components and its lack of cross-component coherence could affect resulting load calculations. This issue with the standard KSEC model is not directly addressed in this paper, but the authors acknowledge its importance and further discussion is given to the topic in Sec. 4.

The simulation procedure for the proposed constrained KSEC method is given as follows. Consider a set of given, measured turbulence time series

$$\mathrm{C} = \{x_{c1}(\boldsymbol{x}_{c1}, t), \ldots, x_{cn_c}(\boldsymbol{x}_{cn_c}, t)\}, \tag{1}$$

where x_{ci} is a measured time series for a particular spatial location $\boldsymbol{x}_{c1}$ and turbulent component, t is time, and n_c is the total number of constraining time series. Note that n_c encompasses both spatial locations and which turbulent component to which the time series corresponds. For example, if the constraining data comes from a met mast with five three-dimensional (3D) sonic anemometers, the number of constraining time series is $n_c = 5 \times 3 = 15$.

Let us assume that we would like to simulate a series of time series

$$\mathrm{S} = \{y_{s1}(\boldsymbol{x}_{s1}, t), \ldots, y_{sn_c}(\boldsymbol{x}_{sn_s}, t)\}, \tag{2}$$

where y_{si} is a simulated time series for a particular turbulent component and spatial location $\boldsymbol{x}_{s1}$ and n_s is the number of points to simulate. Just as in the unconstrained KSEC method, we begin the turbulence simulation in the Fourier domain.

Assemble the Fourier components for all spatial points, both constrained and simulated, as follows:

$$Z(f_k) = [X_{c1}(f_k), \ldots, X_{cn_c}(f_k), Y_{s1}(f_k), \ldots, Y_{sn_s}(f_k)], \tag{3}$$

where a capital letter indicates a component in the Fourier domain and f_k is a particular frequency. For notational convenience, we will denote the vectors in Eq. (3) as

$$Z(f_k) = [X(f_k), Y(f_k)]. \tag{4}$$

The values of X are known *a priori*, as these are simply the Fourier transforms of the constraining measured time series. We seek to calculate the values of Y, but it must be done such that the correct coherence relationships with the constraining time series are maintained. Consider, then, the following equation:

$$Z(f_k) = \begin{bmatrix} \vdots \\ X_{ci}(f_k) \\ \vdots \\ Y_{si}(f_k) \\ \vdots \end{bmatrix} = [L(f_k)] \begin{bmatrix} \vdots \\ C_{ci}(f_k) \\ \vdots \\ U_{si}(f_k) \\ \vdots \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} X(f_k) \\ Y(f_k) \end{bmatrix} = [L(f_k)] \begin{bmatrix} C(f_k) \\ U(f_k) \end{bmatrix}. \tag{6}$$

This equation corresponds to the phase-correlation step in the unconstrained KSEC method. Here, $[L(f_k)]$ is the lower-triangular Cholesky decomposition of the covariance matrix $[\Sigma(f_k)]$, which is defined such that

$$\Sigma_{ij}(f_k) = |Z_i(f_k)|\,|Z_j(f_k)|\,\mathsf{Coh}_{ij}(f_k), \tag{7}$$

[1] It should be noted that TurbSim features several other models in addition to the standard KSEC model.

where $\mathsf{Coh}_{ij}(f_k)$ is the theoretical complex coherence between component/location i and component/location j for frequency f_k and $|\cdot|$ represents taking the magnitude. Note that, in the standard KSEC model, $\mathsf{Coh}_{ij}(f_k) = 0$ unless i and j are both longitudinal turbulent components.

The theoretical magnitudes for the simulation points can be specified in any way. In this paper, we consider two simple techniques: 1) generating them according to the Kaimal spectrum or 2) linearly interpolating them by height from the measured time series. Each technique has its benefits and drawbacks. The first technique is not affected by the potential sparsity of measurements that could negatively affect the second technique. For example, if for some reason the measured spectra at the measurement point are very wrong, the entire turbulence box would be affected and could produce inaccurate validation results when used as turbulent inflow to a wind turbine model. However, the second technique guarantees that a simulated time series will converge to a measured time series as the simulation point approaches the measurement point.

Consider, for example, the plots shown in Fig. 1. Both plots were generating by simulating 10 realizations of turbulent time series at spatial locations that were various distances away from the measurement point. The spatial locations converged towards the measurement points in three different manners: laterally (i.e., the simulation point started at the same height but with some horizontal offset), vertically (i.e., the simulation point began a certain distance above the measurement point and converged downward), and from a random direction. For each in-plane separation distance and turbulent realization, two turbulent time series were simulated: one with magnitudes generated from the Kaimal spectrum and one with magnitudes that were linearly interpolated from the measured magnitudes. The resulting average absolute error was calculated as follows

$$AAE = \overline{|u_{sim} - u_{meas}|} \tag{8}$$

and the results for the Kaimal and data magnitudes are shown in the upper and lower plots, respectively.

As expected, there is a bias in the turbulent simulations generated with Kaimal magnitudes that prevents the error from ever converging. In fact, once the simulation and measurement points are within 0.1 m of one another, there is no further decrease in error. This is in sharp contrast with the error for the turbulent simulations generated from the interpolated data magnitudes, which show a consistent decrease in error for all tested in-plane separation distances. The trend is approximately linear in the log-log axes, as indicated by the red line. This consistent decrease in error for the magnitudes interpolated from data, and the lack thereof for the magnitudes generated from the Kaimal spectrum, highlights the importance of picking a model for the simulation magnitudes that converge to the measurement magnitudes as the distance to the measurement point decreases.

The key novel aspect of the constrained KSEC technique proposed in this paper is the creation of the rightmost vector in Eq. (6). In the traditional KSEC method, this vector is generated such that $U_{si}(f_k) = \exp(j2\pi\mathcal{U})$, where $\mathcal{U}$ is a uniformly distributed random variable on $[0, 1)$. This is fine for $U(f_k)$ in the constrained KSEC method, but the elements of $C(f_k)$ cannot be generated in this manner or the correct values of $X(f_k)$ will not be recovered. To address this issue, we separate $[L(f_k)]$ into blocks as follows:

$$\begin{bmatrix} X(f_k) \\ Y(f_k) \end{bmatrix} = \begin{bmatrix} L_1(f_k) & 0 \\ A(f_k) & L_2(f_k) \end{bmatrix} \begin{bmatrix} C(f_k) \\ U(f_k) \end{bmatrix}, \tag{9}$$

where $[L_1(f_k)]$ and $[L_2(f_k)]$ are lower-triangular matrices, $[0]$ is a matrix of zeros, and $[A(f_k)]$ is a fully-populated matrix. Matrix $[L(f_k)]$ is invertible because it is a positive-definite covariance matrix by nature, in which case $[L_1(f_k)]$ is also invertible due to the nature of lower-triangular matrices. Therefore,

$$C(f_k) = [L_1(f_k)]^{-1} X(f_k), \tag{10}$$

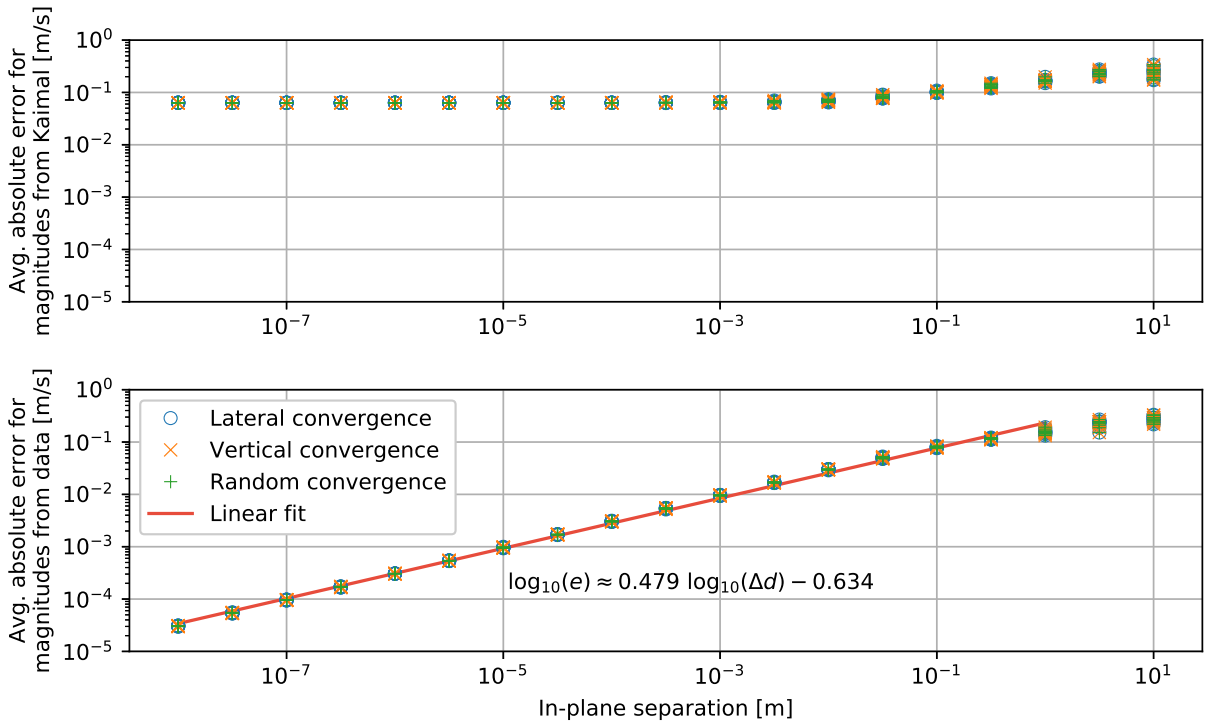

Figure 1. Average absolute error between a simulated time series and a measured time series as the simulation location converges to the measurement location laterally, vertically, and from a random direction. The top plot used turbulence generated with magnitudes from the Kaimal spectrum, and the bottom plot used magnitudes lineraly interpolated from the measurement data. Ten realizations were generated for each separation distance and are overlaid in the plots.

in which case

$$Y(f_k) = [A(f_k)][L_1(f_k)]^{-1}X(f_k) + [L_2(f_k)]U(f_k). \tag{11}$$

The proposed constrained turbulence simulation algorithm is summarized in Alg. 1.

Algorithm 1 Pseudocode for proposed constrained KSEC algorithm

for each Fourier frequency f_k **do**
  compute covariance matrix $[\Sigma(f_k)]$
  apply Cholesky decomposition to generate $[L(f_k)]$
  compute $C(f_k)$ according to Eq. (10)
  sample $U(f_k)$ randomly
  compute $Y(f_k)$ according to Eq. (11)
end for

2.2. Computational implementation

The proposed constrained turbulence methodology has been implemented in an open-source Python package called PyConTurb, which stands for "Python Constrained Turbulence". The package is hosted on GitLab (https://gitlab.windenergy.dtu.dk/rink/pyconturb), and the source code, installation instructions, and code documentation can be found at the link. The algorithm

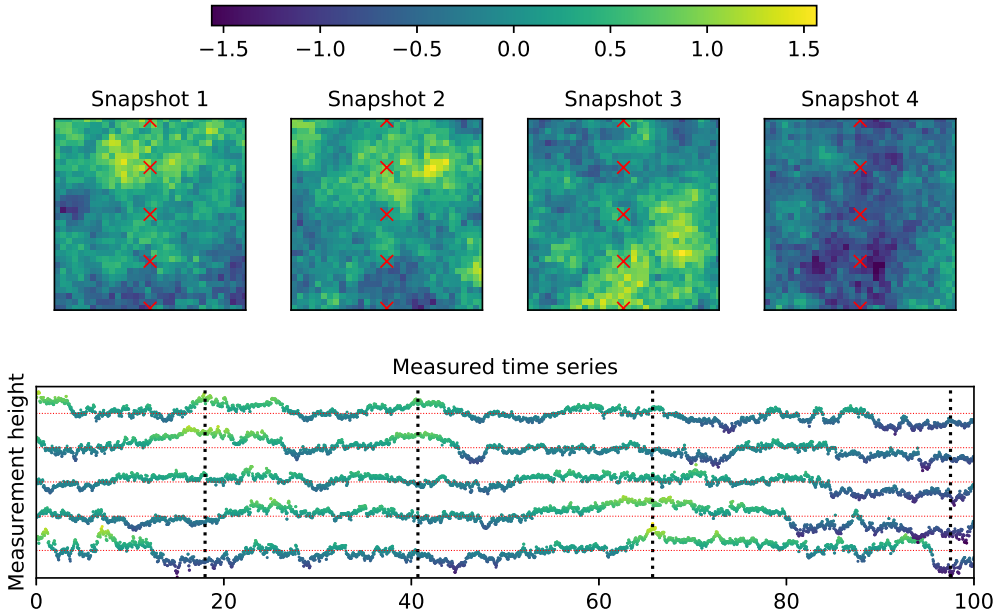

Figure 2. Snapshots of a turbulence box generated from the five measured time series shown in the bottom subplot. The measured time series are from a met mast installed at DTU Risø campus. The dotted lines in the bottom subplot indicate the timepoints at which the four snapshots were taken, and the red x's in the snapshot plots indicate the measurement locations. The wind speed units are m/s.

was modified slightly to improve performance, and notes on those modifications are also in the linked repository.

3. Case study: generating turbulence boxes from met mast data

In this section, PyConTurb is used in conjunction with met mast data recorded at DTU Risø campus to demonstrate its utility with real measurements. The data were recorded using five 3D sonic anemometers at different heights (18 m, 31 m, 44 m, 57 m, and 70 m). Only the longitudinal turbulence components were used to constrain the simulations because the standard exponential coherence model has no spatial coherence for the lateral and vertical turbulence components. The generated turbulent grid was a 32×32 square grid that was 53 m wide/tall and centered at 44 m. The simulation time step was 1/35 s, which matched the measurements.

A comparison of snapshots from the simulated constrained turbulence box is shown in Fig. 2. The measured time series are plotted in the bottom axes, and the black dotted lines indicate the times at which the snapshots were taken. The red x's on the upper snapshot plots indicate the measurement points. The longitudinal wind speed values in all plots are colored according to the colorbar at the top of the figure. The snapshots were chosen specifically to highlight the largest increases/decreases in the measured turbulent time series.

As expected, the snapshots display coherent increases/decreases in the turbulent field near the corresponding measurement point featuring the increase/decrease. For example, Snapshot 3 was chosen to highlight the significant increase in wind speed at the 18-m measurement, and there is a corresponding area of increased wind speeds near that measurement point in the corresponding snapshot. Similarly, Snapshot 4 was chosen to highlight the decreased wind speeds for all measurement points at that time stamp, and a similar decrease in wind speed is

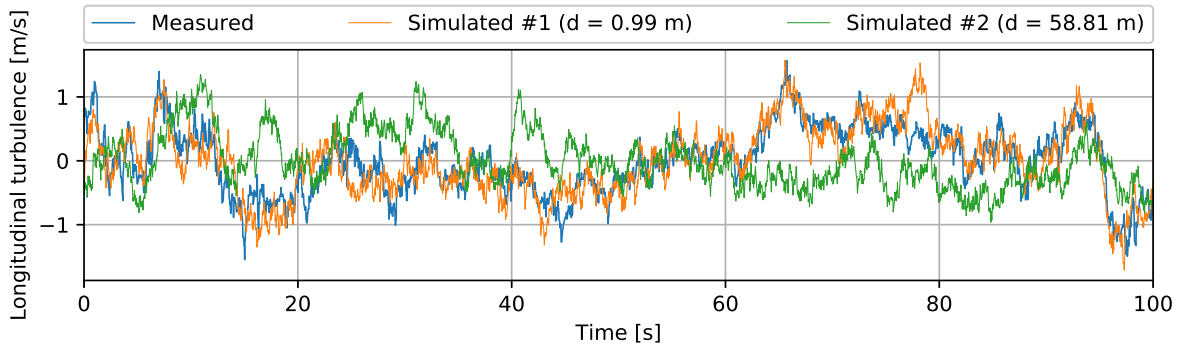

Figure 3. Comparison of measured met mast time series (18 m) to two simulated time series taken from a turbulence box generated from the met mast data. The orange simulated series is spatially close to a measurement point (0.99 m in-plane separation), whereas the green simulated time series is further away (58.81 m in-plane separation).

clearly reflected in the corresponding snapshot.

The simulated turbulence's ability to "track" the measured time series is further demonstrated in Fig 3. Three turbulent time series are overlaid in the plot: the measured time series at 18 m (blue), a simulated time series 0.99 m away (orange), and a simulated time series 58.81 m away (green). It is apparent that, as expected, the distant simulated time series shown in green does not follow the same trends as the measured time series. On the other hand, the close simulated time series shown in orange has very similar trends to the measured time series, including the sudden peak in wind speed near 65 s. This tracking ability is essential for one-to-one wind turbine validation, when the largest fatigue loads can be caused by single peaks in a measured wind record.

The fact that the simulated turbulence mimics the measured turbulence in the time domain is a direct result of the spatial coherence between the two points. We also verified that the turbulent time series generated using PyConTurb have the desired spatial coherence relationships with the measured time series using 100 turbulence realizations 1 m away from the measured time series. The results for the spatial coherence estimated from the 100 realizations are shown in Fig. 4. For the smaller coherence values, there are some deviations between the theoretical coherence (blue line) and estimated coherence (orange circles) due to the small number of turbulent realizations used to calculate the coherence. For the larger coherence values, however, there is excellent agreement between the theoretical and estimated coherence. Thus, PyConTurb can easily simulate turbulence boxes from real data measured using 3D sonic anemometers on a met mast and maintain the correct spatial coherence relationships with the measured data.

4. Discussion

While we have clearly demonstrated PyConTurb's ability to simulate turbulence boxes that are constrained by real measurements, there are a few points of interest that merit further discussion.

First, as mentioned above, for this paper we simply linearly interpolate the Fourier magnitudes according by height above the ground to obtain the simulation magnitudes. The rationalization for using this technique is that the standard Kaimal spectrum is also a function of height. However, this technique is perhaps not very reflective of the conditions found in real atmospheric turbulence. Moreover, it makes the entire simulated turbulence box very sensitive to the measured data if only one or two measurements points are used. Other, more sophisticated techniques to generate the simulation magnitudes from the measured magnitudes

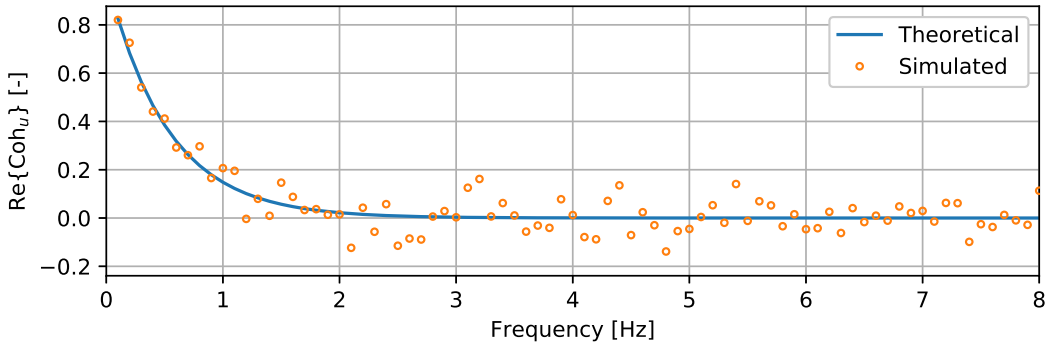

Figure 4. Theoretical and estimated spatial coherence between the 44-m met mast measurement and a simulation point 1 m away

will be investigated in the future and implemented into PyConTurb.

Second, for the time being, PyConTurb utilizes the standard exponential coherence model specified in IEC 61400-1 Ed. 3. As noted in Sec. 2.1, this is an inaccurate model due to its exclusion of spatial coherence in the lateral and vertical turbulent components. In light of this, PyConTurb was intentionally given a modular structure to allow the specification and easy investigation and comparison of new, improved spatial coherence models for wind energy applications. Thus, although the utilization here of the standard exponential coherence could be considered a drawback of the paper, it is important to note that this paper is only intended to present PyConTurb and its abilities. Future work is already underway to improve the coherence model in order to produce better one-to-one wind turbine validation results.

Third and last, it is important to emphasize the flexibility of the constraining data. We have here demonstrated PyConTurb's abilities using met mast data, but this is by no means the only data that can be used to generated turbulence boxes. For example, even a single measurement from one cup anemometer could be used to simulate a turbulence box. Alternatively, a spatial field of longitudinal turbulence measured with a spinner lidar could be used to generate the lateral and vertical turbulence components. Thus, PyConTurb is an extremely flexible tool that can be used to simulate constrained turbulence for a wide variety of data sources.

5. Conclusions

This paper presents 1) a novel method to simulate turbulence that is constrained by measured time series for one-to-one wind turbine validation and 2) the open-source tool into which the method it has been implemented, called "PyConTurb". The method is based on the same technique used in the KSEC model in wind turbine design standard IEC 61400-1 Ed. 3, but it includes the measured time series in the spatial correlation step to ensure that the simulated turbulence box follows the same trends as nearby measurement points. If a simulation point and measurement point are collocated, the simulated time series perfectly recreates the measured time series.

The presented technique was implemented in Python, and the PyConTurb package is readily available on GitLab for installation. The utility of PyConTurb for real-measurement applications was demonstrated using time series measured from 3D sonic aneometers on a met mast at DTU Risø campus. Snapshots and time series of the generated turbulence box were compared with the measured time series. As desired, close simulation points "tracked" the measured data well. Additionally, the desired spatial coherence between a simulation point and a measurement point was also recovered. Thus, PyConTurb was shown to effectively simulate turbulence boxes

that are constrained by measured data, which is extremely useful in one-to-one wind turbine validation studies. More importantly, PyConTurb can be used for a very wide variety of input data (3D sonic anemometers, cup anemometers, spinner lidars, etc.) for future wind turbine validation activities.

References

[1] B. J. Jonkman and L. Kilcher, "TurbSim user's guide: Version 1.06.00," National Renewable Energy Laboratory, Tech. Rep., September 2012.
[2] M. Nielsen, G. C. Larsen, J. Mann, S. Ott, K. S. Hansen, and B. J. Pedersen, "Wind simulation for extreme and fatigue loads," Risø National Laboratory, Roskilde, Denmark, Tech. Rep. Risø-R-1437(EN), January 2004.
[3] N. Dimitrov and A. Natarajan, "Application of simulated lidar scanning patterns to constrained gaussian turbulence fields for load validation," *Wind Energy*, vol. 20, no. 1, pp. 79–95, may 2016.
[4] J. Mann, "The spatial structure of neutral atmospheric surface-layer turbulence," *Journal of Fluid Mechanics*, vol. 273, pp. 141–168, 1994.
[5] International Electrotechnical Commission, *Wind turbines - part 1: design requirements*, International Electrotechnical Commission Std. 61 400-1, 2005, 3rd ed.